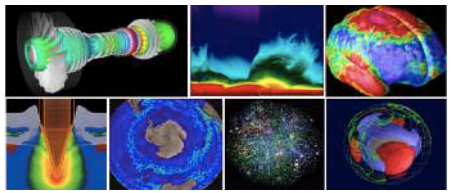
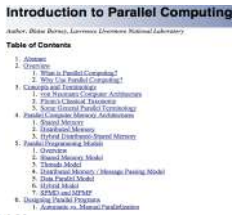
Tính toán song song và phân tán

PGS.TS. Trần Văn Lăng
langtv@vast. vn

Tài liệu: Introduction to Parallel Computing

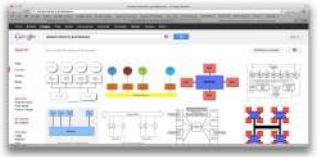
Blaise Barney, Lawrence Livermore National Laboratory
https://computing.llnl.gov/tutorials/parallel_comp/





5. Thiết kế chương trình song song

1. Song song tự động so với song song bằng tay
2. Hiểu được bài toán và chương trình
3. Phân hoạch
4. Truyền thông
5. Tích tụ
6. Ánh xạ



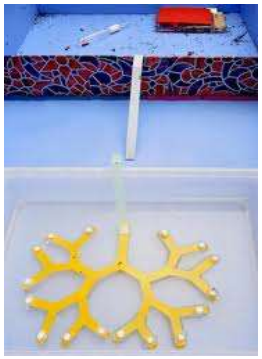
Tài liệu: Designing and Building Parallel Programs

Ian. Foster, Addison-Wesley, 1995, http://www.mcs.anl.gov/dbpp

6. Sự phụ thuộc dữ liệu
7. Cân bằng tải
8. Mức độ chi tiết
9. Nhập xuất
10. Chi phí của lập trình song song
11. Phân tích hiệu năng

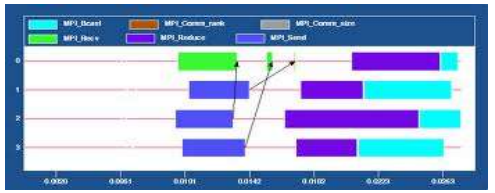
5.1 Việc song song hóa

- Thiết kế và phát triển các chương trình song song có đặc trưng đó là một quá trình rất thủ công.
- Người lập trình thường là phải chịu trách nhiệm cho cả việc nhận ra và hiện thực song song trong chương trình.



- Việc viết chương trình song song bằng tay thường mất nhiều thời gian, phức tạp hay bị lỗi
- Trong vài năm gần đây, có xu hướng phát triển công cụ hỗ trợ người lập trình chuyển đổi một chương trình tuần tự sang song song như là sự tiền xử lý. Tuy nhiên, có nhiều vấn đề còn bỏ ngỏ

- Một trình biên dịch song song thường làm việc theo hai cách khác nhau:
  - Tự động hòan toàn
  - Theo sự hướng dẫn của người lập trình



Tự động hoàn toàn

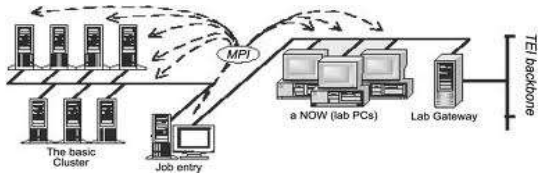
- Trình biên dịch phân tích mã nguồn và xác định cơ hội song song.
- Phân tích này bào gồm việc tìm ra các phần khó có cơ hội song song, cũng như những trọng số có thể để cải thiện hiệu năng.
- Các vòng lặp (do, for) là đối tượng xem xét thường xuyên nhất khi cần song song tự động.

Theo sự hướng dẫn

- Sử dụng trình biên dịch có hướng dẫn người lập trình; qua đó chỉ ra một cách rõ ràng cách thức để song song một đoạn chương trình
- Cũng có thể sử dụng kết hợp với hệ thống tự động.

- Nếu chúng ta bắt đầu với một mã tuần tự với thời gian cũng như chi phí hạn chế, thì việc dùng hệ thống song song tự động là một chọn lựa.
- Tuy nhiên, cũng có một số khác biệt quan trọng khi áp dụng song song tự động:
  - Đưa ra kết quả sai
  - Hiệu năng bị gảm

  - Song song bằng thủ công linh hoạt hơn
  - Bị hạn chế trong một số đoạn lệnh (hầu hết là vòng lặp)
  - Đoạn mã có thể không cần song song.



5.2 Bài toán và chương trình song song

- Bước đầu tiên trong phát triển phần mềm song song là hiểu được bài toán muốn giải theo cách song song.
- Nếu có một chương trình tuần tự, cần phải hiểu rõ mã nguồn của nó.
- Trước khi dành thời gian để phát triển một giải pháp song song cho bài toán, hãy xác định bài toán có song song được hay không.

Bài toán song song được

- Ví dụ cần tính toán thế năng của hàng ngàn cấu trúc phân tử độc lập. Từ đó tìm thế năng cực tiểu.
  - Bài toán song song được, bởi thế năng từng cấu trúc phân tử có thể tính độc lập.
  - Bài toán tìm cực tiểu cũng có thể song song được bằng cách xem xét từng khúc đã tính.

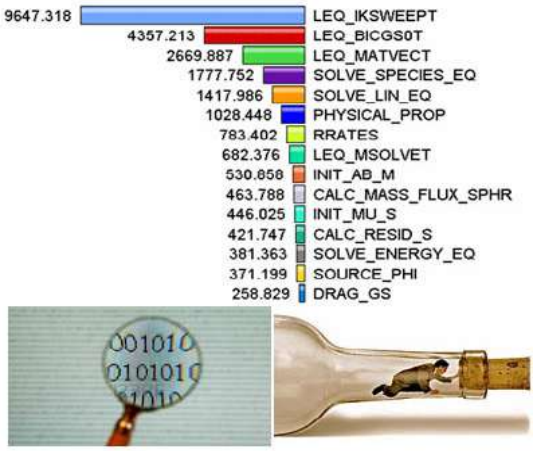
Bài toán không song song được

- Ví dụ, tính toán dãy số Finonacci (0, 1, 1, 2, 3, 5, 8, 13, 21, ...) theo công thức F(n) = F(n-1) + F(n-2)
- Đây là bài toán không song song được vì một phần tử của dãy số được tạo ra phụ thuộc vào các phần tử tạo ra trước đó.
  - Cụ thể, F(n) phụ thuộc vào F(n-1) và F(n-2)

Sau đó, nếu song song được

- Xác định các điểm nóng của chương trình
  - Ở đâu là công việc thực hiện
  - Trong các chương trình tính toán khoa học và kỹ thuật, công việc chính chỉ ở một vài nơi.
  - Sử dụng công cụ phân tích để hỗ trợ cho việc xác định này.
  - Bỏ qua những phần của chương trình mà ít sử dụng năng lực của CPU

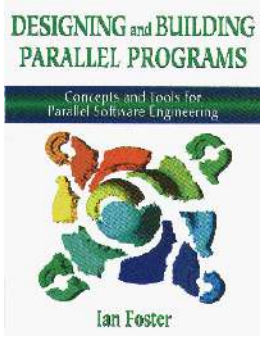
- Xác định điểm thắt cổ chai, điểm tắt nghẽn (***bottlenecks***) trong chương trình
  - Đó là những phần thực hiện với thời gian không cân đối với các phần khác. Từ đó làm phá hủy cơ hội song song. Chẳng hạn, việc nhập xuất thường diễn ra chậm.
  - Có thể cơ cấu lại chương trình, hoặc sử dụng thuật toán khác để loại bỏ vùng thực hiện chậm này.



- Xác định những yếu tố cản trở viêc song song. Đa phần sự cản trở này là do sự phụ thuộc dữ liệu (*data dependence*). Chẳng hạn, dãy số Fibonacci.
- Khảo sát các thuật toán khác nếu có thể. Đây là sự xem xét quan trọng nhất khi thiết kế ứng dụng song song.
- Tận dụng lợi thế của các phần mềm song song cũng như những thư viện toán học hiệu quả của các nhà cung cấp.

Đặc điểm của chương trình song song

- Đồng thời (concurrency)
- Khả năng điều chỉnh (scalability)
- Địa phương (locality)
- Mô đun (modularity)

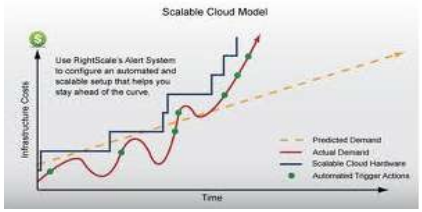


Concurrency, scalability

- Tính đồng thời (concurrency), nhằm để có thể thực hiện trên nhiều bộ xử lý.
- Tính điều chỉnh được (scalability) thể hiện việc thuật giải có thể cài đặt mà không quan tâm đến số bộ xử lý mà chúng sẽ thi hành.

Scalability

- Khả năng điều chỉnh thể hiện qua việc, khi số bộ xử lý tăng lên, số tiến trình trên một bộ xử lý sẽ thu ít lại, nhưng thuật giải chính nó lại không cần thay đổi.



Locality

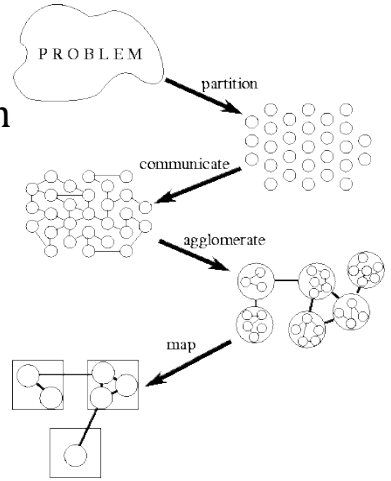
- Khi tính địa phương (locality) nhằm giảm chi phí thời gian của thuật giải.
  - Do khi đó việc truy cập đến local data (dữ liệu trên máy tại chỗ) xẩy ra thường xuyên hơn việc truy cập đến những remote data (dữ liệu ở xa)

Modularity

- Tính mô đun hoá (modularity), hoàn toàn giống với sự mong đợi trong lập trình tuần tự.
  - Thực chất là sự phân hoạch những thực thể phức tạp thành các thành phần đơn giản hơn.

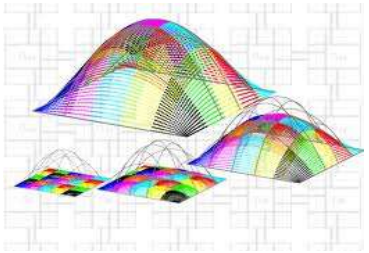
5.3 Phân chia (Partitioning)

- Có 4 giai đoạn cần thiết trong việc thiết kế một chương trình song song
  - Phân chia (Partitioning)
  - Giao tiếp (Communication)
  - Tích tụ (Agglomeration)
  - Ánh xạ (Mapping)



- Bước đầu tiên của việc thiết kế chương trình song song là tách bài toán thành ra các khúc, các phần công việc rời rạc để phân phối như là task.
- Công việc này được biết như là sự phân rã (decomposition) hoặc phân chia (partitioning).

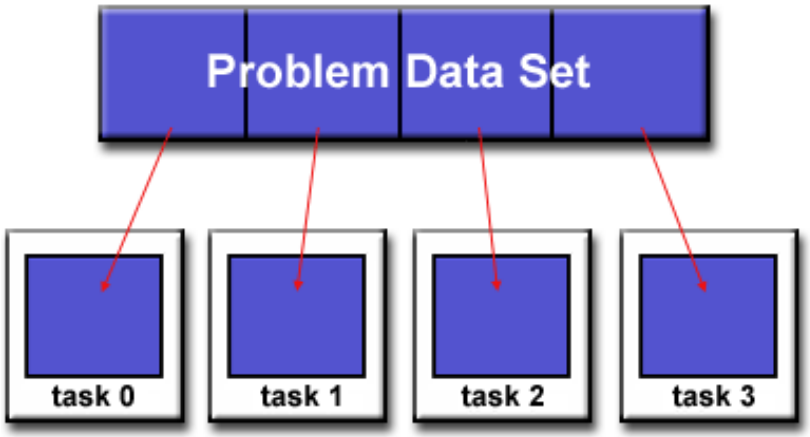
- Đây là giai đoạn tạo cơ hội song song,
- Khi phân rã, việc tính toán được thực hiện trên những vùng dữ liệu nhỏ hơn, và các hành động cũng được đưa về thành những tiến trình nhỏ hơn.



- Có 2 cách để phân chia công việc tính toán cho các task song song:
  - Phân rã theo miền (***domain decomposition***)
  - Phân rã theo chức năng (***functional decomposition***)

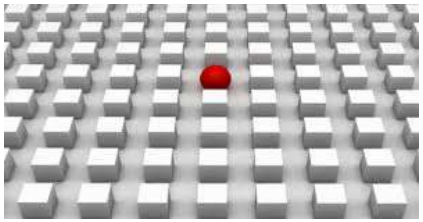
Phân rã theo miền

- Khi thiết kế chương trình, người lập trình trước hết thường quan tâm đến dữ liệu liên kết với bài toán
- Nên trong loại phân rã này, dữ liệu liên quan đến bài toán được phân rã; để rồi mỗi task song song làm việc trên một phần của dữ liệu.

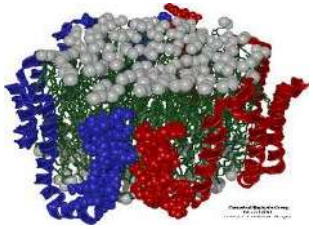


- Thông thường việc phân rã theo miền được căn cứ vào:
  - Dữ liệu đầu vào của chương trình,
  - Kết quả đầu ra được tính toán.
  - Những dữ liệu lưu giữ giá trị trung gian quá trình thực hiện.

- Phân rã theo dữ liệu là một phương pháp phân ly hiệu quả và được sử dụng nhiều nhất trong việc xác định tính đồng thời của thuật toán để có thể thao tác trên các cấu trúc dữ liệu lớn.



- Phương pháp này bao gồm 2 bước:
  - Dữ liệu trong bước tính toán sẽ được phân ra thành từng phần
  - Phần dữ liệu này sẽ được chuyển cho các task để thực thi.



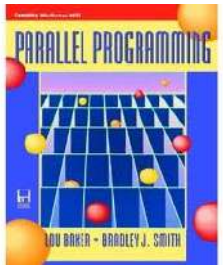
Ví dụ nhân ma trận

- Nhân 2 ma trận A và B, kết quả trả về là ma trận C. Giả sử ma trận A và B đều có kích thước n x n.
- Trước tiên phân từng ma trận thành 4 khối hay 4 ma trận con, bằng cách chia đôi các chiều của ma trận.
- Bốn ma trận con của ma trận kết quả C - mỗi phần có kích thước n/2 x n/2 - có thể được tính toán độc lập với nhau bởi 4 tiến trình.

- Bước 1: chia ma trận nhập và ma trận xuất thành 2 x 2 ma trận con.

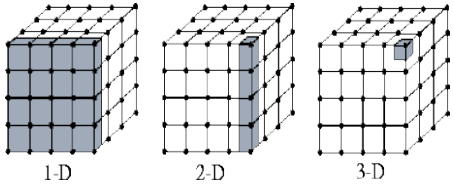
$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

- Bước 2: Sự phân chia việc nhân ma trận A thành 4 task dựa trên việc chia ma trận của bước 1:
  - Task 1: $C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$
  - Task 2: $C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{22}$
  - Task 3: $C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$
  - Task 4: $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$

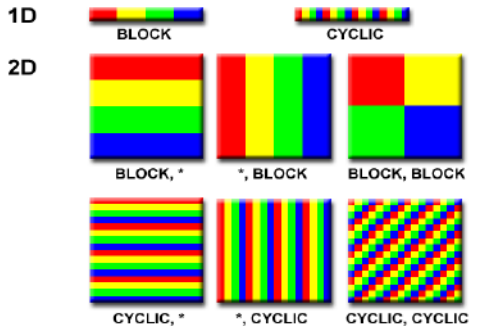


Ví du: chia miền trong lưới tính toán

- Phân rã miền cho bài toán đòi hỏi lưới 3 chiều.
- Khi đó miền phân chia là
  - 1-D, mỗi task bao gồm 5 x 5 điểm,
  - 2-D, mỗi task bao gồm 5 điểm,
  - 3-D, mỗi task chỉ có 1 điểm nút.

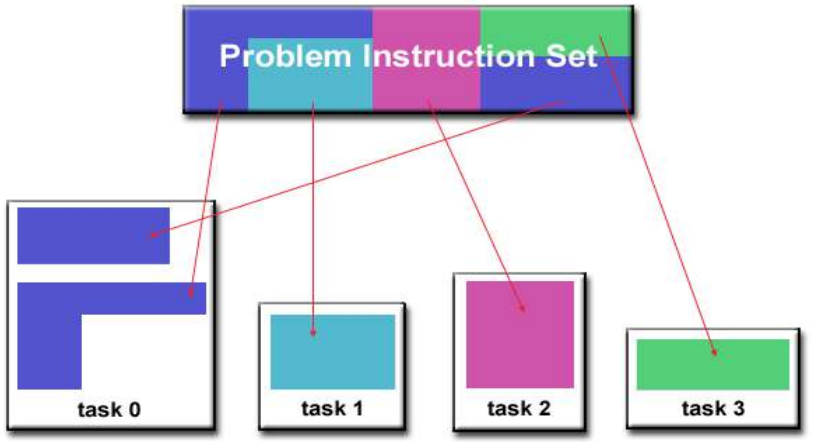


- Trường hợp 3-D được coi là mềm dẽo nhất và dễ dàng chấp nhận trong giai đoạn thiết kế



Phân rã theo chức năng

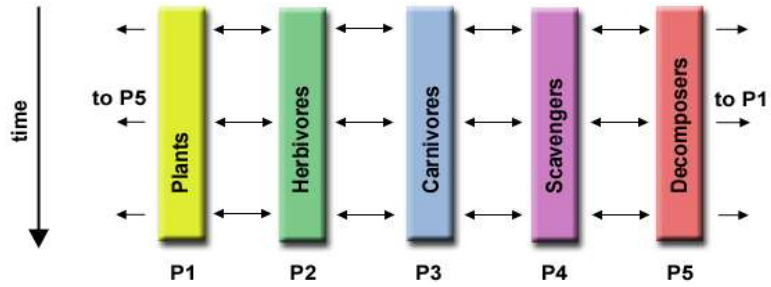
- Sự phân rã tốt ở đây không chỉ dừng lại ở việc phân rã dữ liệu tính toán, mà còn cả việc phân chia các thao tác tác động tới dữ liệu.
- Theo cách tiếp cận này, trọng tâm là những tính toán, không phải là dữ liệu.



Mô hình hệ sinh thái

- Mỗi chương trình tính toán quần thể của một nhóm nào đó, mà ở đó sự tăng trưởng của mỗi nhóm phụ thuộc vào nhóm lân cận.
- Như một sự tiến triển, mỗi tiến trình tính toán trạng thái hiện tại, rồi trao đổi thông tin với quần thể bên cạnh.
- Tất cả các task tiến triển lên để tính toán trạng thái ở bước thời gian tiếp theo.

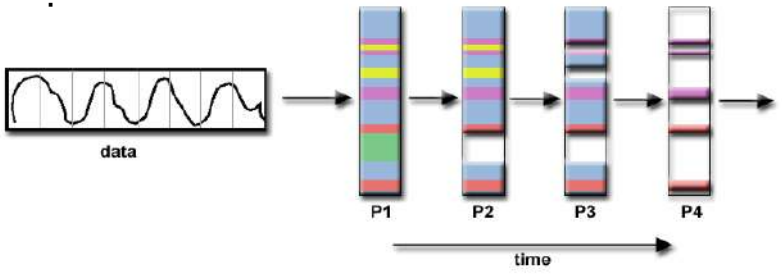
- Xét các nhóm: thực vật, động vật ăn cỏ, động vật ăn thịt, động vật ăn xác chết, sinh vật phân hủy



Xử lý tín hiệu

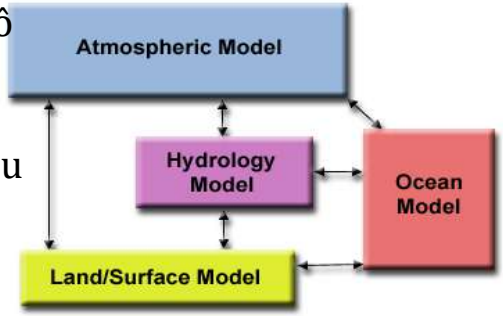
- Một tập hợp dữ liệu tín hiệu audio được chuyển qua 4 bộ lọc tính toán phân biệt với nhau.
- Mỗi bộ lọc là một tiến trình riêng biệt.
- Các phân đoạn đầu của dữ liệu phải chuyển qua bộ lọc đầu tiên trước khi tiến đến bộ lọc thứ hai. Khi nó làm việc, phân khúc thứ hai của dữ liệu được chuyển qua bộ lọc thứ nhất.

- Theo thời gian, phân khúc thứ tư của dữ liệu ở trong bộ lọc thứ nhất, như vậy tất cả 4 task đều bận rộn.



Mô hình khí hậu

- Mỗi thành phần của mô hình là một task riêng biết. Mũi tên đại diện cho việc trao đổi dữ liệu giữa các thành phần trong suốt quá trình tính toán.



- Mô hình khí quyển (atmosphere model) sinh ra dữ liệu vận tốc gió để sử dụng trong mô hình đại dương (ocean model)
- Mô hình đại dương sinh ra dữ liệu nhiệt độ bề mặt biển dùng cho mô hình khí quyển, mô hình thủy lực (hydrology model) và mô hình mặt đất (land/surface model), v.v...

Lưu ý về việc phân rã

- Ngoài việc phân rã theo 2 cách riêng biệt, trong quá trình phân chia để được các task song song, còn sử dụng kết hợp cả 2 phương pháp.
  - Chẳng hạn, Sau khi đã chia tính toán (phân rã chức năng) thành những tiến trình rời, có thể tiếp tụctìm ra dữ liệu cần cho những tiến trình này.
  - Những dữ liệu này cũng có thể tách rời ra bởi việc phân rã theo miền.

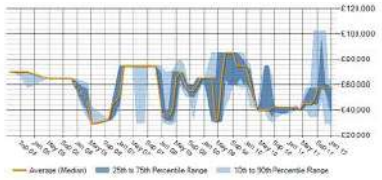
5.4 Truyền thông (giao tiếp)

- Là sự liên lạc qua lại,
- Đây chính là sự phối hợp giữa các task để có được sự hoạt động phù hợp.
- Những task sinh ra bởi sự phân rã được dùng để thi hành đồng thời.

- Tuy nhiên, trong trường hợp tổng quát những tiến trình này không thể thi hành một cách độc lập; đòi hỏi dữ liệu liên kết với tiến trình khác.
- Khi đó, dữ liệu được truyền giữa những tiến trình, các tính toán mới được tiếp tục.
- Chính vì vậy, sự giao tiếp như một giai đoạn cần thiết trong việc thiết kế thuật giải song song.

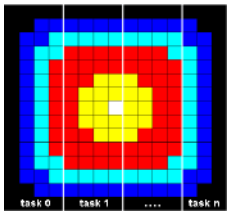
- Trong việc phân rã theo miền, những đòi hỏi về sự giao tiếp khó có thể xác định.
- Khi đó việc truyền dữ liệu cần phải quản lý chặt chẽ để bảo đảm sự hoạt động của tiến trình

- Ngược lại, sự giao tiếp đòi hỏi trong những thuật giải nhận được từ phân rã chức năng thường không khó khăn lắm.
- Bởi, thông thường chúng tương ứng với dòng dữ liệu truyền giữa các tiến trình.

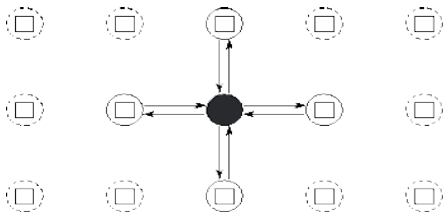


Ví dụ: bài toán lan truyền nhiệt

- Trong bài toán lan truyền nhiệt trên một mặt phẳng, nhiệt độ của một điểm được tính toán dựa trên nhiệt độ của điểm bên cạnh.
- Từ đó, nếu bên cạnh do một task khác tính toán, thì phải truyền dữ liệu nhiệt độ này giữa các task.



- Khi rời rạc hóa để tính toán trên máy, đưa vào lưới sai phân. Chẳng hạn với lưới sai phân hữu hạn gồm 5 điểm như:

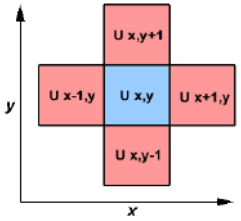


- Khi đó, để tính toán giá trị tại nút (i,j) ở thời điểm t+1 chúng ta cần dựa vào giá trị tại nút đó và 4 nút bên cạnh vào thời điểm thứ t theo công thức:

$$X_{ij}^{(t+1)} = \frac{4X_{ij}^{(t)} + X_{i-1j}^{(t)} + X_{i+1j}^{(t)} + X_{ij-1}^{(t)} + X_{ij+1}^{(t)}}{8}$$

- Với sự phân rã theo miền, mỗi tiến trình tại điểm lưới (*i,j*) phải tính toán dãy các giá trị *X*.

$$X_{ij}^{(1)}, X_{ij}^{(2)}, X_{ij}^{(3)}, \ldots$$



- Việc tính toán các giá trị *X* này đòi hỏi giá trị của 4 dãy tại các nút lân cận:

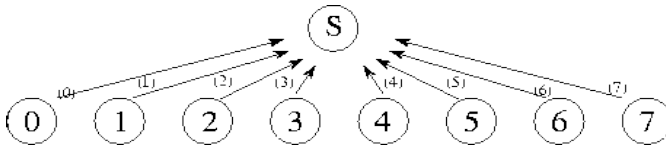
$$X_{i-1\,j}^{(0)}, X_{i-1\,j}^{(1)}, X_{i-1\,j}^{(2)}, X_{i-1\,j}^{(3)}, \cdots$$
$$X_{i+1\,j}^{(0)}, X_{i+1\,j}^{(1)}, X_{i+1\,j}^{(2)}, X_{i+1\,j}^{(3)}, \cdots$$
$$X_{ij-1}^{(0)}, X_{ij-1}^{(1)}, X_{ij-1}^{(2)}, X_{ij-1}^{(3)}, \cdots$$
$$X_{ij+1}^{(0)}, X_{ij+1}^{(1)}, X_{ij+1}^{(2)}, X_{ij+1}^{(3)}, \cdots$$

- Thuật giải trong trường hợp này có thể viết:

```
for t = 0 to T–1 do
   send Xij to each neighbor
   receive Xi–1j,Xi+1j,Xij–1,Xij+1 from neighbors
   compute Xij
endFor
```

Ví dụ: Tính tổng

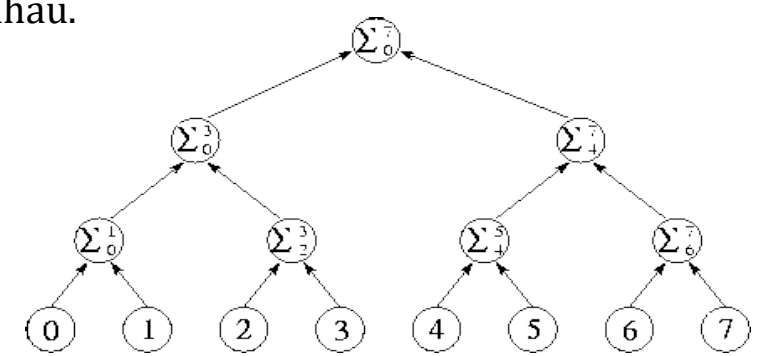
- Giả sử thuật toán tính tổng được phân rã thành các task như hình:



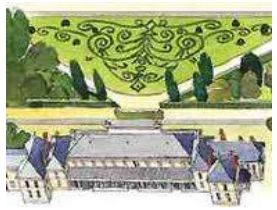
- Như vậy, task S phải giao tiếp với các task từ 0 đến 7. Còn những task này không giao tiếp với nhau.

- Hoặc với tổng $S_i = X_i + S_{i+1}$ trên task thứ *i.*
- Suy ra, S_0 lưu trữ tổng của dãy
- Điều đó cho thấy task thứ *i* giao tiếp với task thứ *i*+1 bên phải của nó.

- Hoặc chia bài toán thành hai hoặc nhiều bài toán đơn giản hơn mà kích thước hầu như xấp xỉ nhau.

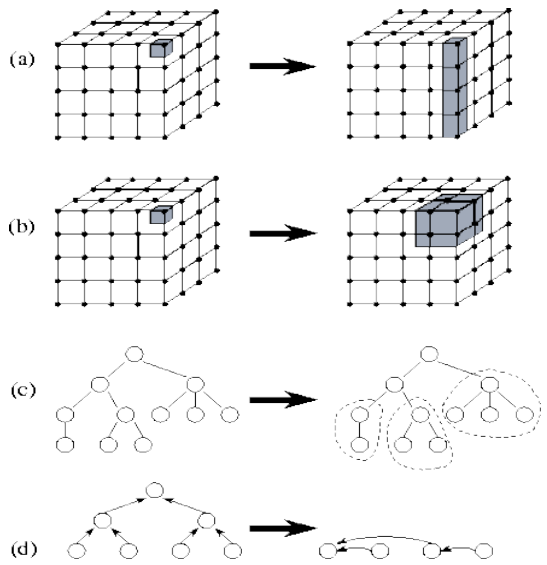


- Giao tiếp giữa các task không nhiều, các task có thể thực hiện đồng thời bởi sự phụ thuộc dữ liệu ít.



5.5 Agglomeration

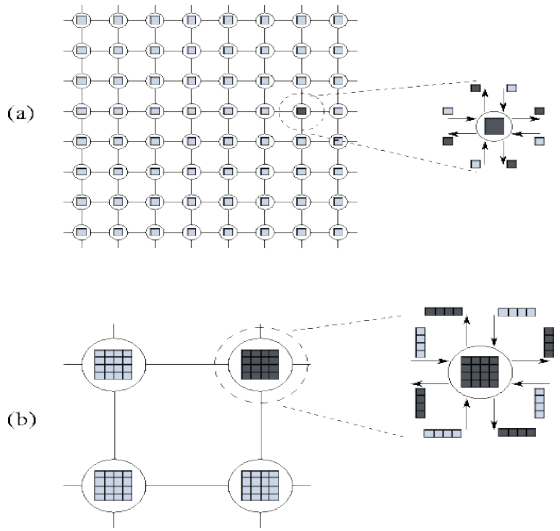
- Đây là giai đoạn gom các task lại để tăng tính địa phương và giảm chi phí giao tiếp.
- Chẳng hạn, thay vì tách thành các miền con 3-D; chỉ tách thành miền con 2-D, hoặc 1-D
  - Khi đó mỗi tiến trình có thể bao gồm nhiều nút hơn.
- Hoặc các cây con có thể được nhóm lại.



Ví dụ: Bài toán lan truyền nhiệt

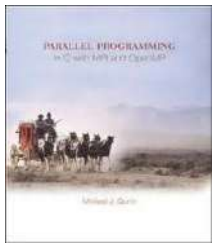
- Một task bao gồm 1 điểm sai phân, có thể tích tụ lại để có một task gồm 16 điểm sai phân.
- Khi đó dữ liệu được truyền giảm từ 256 xuống còn 64.
- Như vậy, số lượng giao tiếp thực hiện trên một đơn vị tính toán sẽ giảm đi.

- Cụ thể như sau:
- Xét 2 trường hợp phân rã mịn và thô như hình a và b.



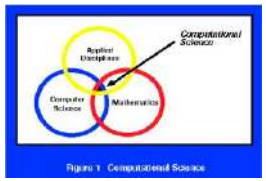
- Hình a. Sự tính toán trên lưới 8 x 8 được phân thành 8 x 8 = 64 task, mỗi task chịu trách nhiệm 1 điểm.
- Hình b. Với cùng tính toán được phân thành 2 x 2 = 4 task, mỗi task chịu trách nhiệm 16 điểm.

- Trong cả hai trường hợp, một task đều giao tiếp với 4 task xung quanh.
- Số dữ liệu được truyền:
  a) là 64 x 4 giao tiếp x 1 = 256 dữ liệu.
  b) chỉ có 4 x 4 giao tiếp x 4 = 64 dữ liệu.



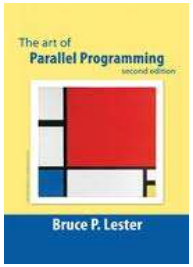
5.6 Mapping

- Khi xây dựng thuật giải song song, yếu tố cơ bản được quan tâm đến đó là số tiến trình ta quen gọi là số task hay số tac vụ.
- Số lượng tiến trình của các bài toán khác nhau hoàn toàn không giống nhau.



- Mục tiêu là cần phải mapping (ánh xạ) sao cho tổng thời gian thi hành đạt cực tiểu.
- Tiêu chí:
  - Các tiến trình có thể thực thi đồng thời đặt trên *những bộ xử lý khác nhau* để tăng khả năng song song.
  - Các tiến trình thường trao đổi với nhau đặt trên *cùng một bộ xử lý* để tăng tính địa phương, giảm chi phí.

- Có hai kiểu ánh xạ
  - Static mapping
  - Dynamic mapping



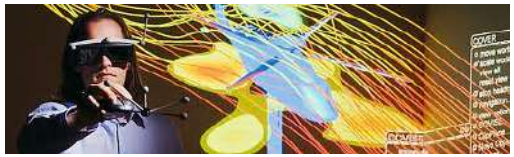
Static Mapping

- Phân phối các tiến trình đến các bộ xử lý trước khi thực thi thuật toán.
- Đối với các tiến trình được tạo ra một cách tĩnh thì ánh xạ động hay ánh xạ tĩnh đều có thể được dùng.

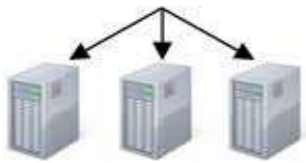
Dynamic Mapping

- Ánh xạ động cần thiết trong các trường hợp khi static mapping có thể gây ra việc phân phối công việc không cân bằng giữa các bộ xử lý.
- Kỹ thuật ánh xạ động phân phối các công việc cho các bộ xử lý trong suốt quá trình thuật toán thực thi.

- Nếu các tiến trình được tạo ra một cách động thì nó sẽ phải được ánh xạ động.
- Nếu các kích cỡ các tiến trình là không biết trước, việc sử dụng static mapping có thể gây nên hiện tượng không cân bằng tải.



- Nếu lượng dữ liệu trong các tiến trình lớn, khi đó ánh xạ động sẽ làm tăng sự di chuyển của dữ liệu giữa các tiến trình, nên static mapping trong trường hợp này sẽ hiệu quả hơn.



5.7 Sự phụ thuộc dữ liệu

- Sự phụ thuộc dữ liệu xẫy ra khi:
  - Thứ tự thực hiện câu lệnh ảnh hưởng đến kết quả của chương trình.
  - Một vùng chứa dữ liệu trong bộ nhớ được dùng nhiều lần bởi nhiều task khác nhau.
- Xem xét sự phụ thuộc là một yếu tố cần thiết, bởi đó là lý do kìm hãm cơ hội sonh song của thuật toán.

Ví dụ

- Xét một vòng lặp về sự phụ thuộc dữ liệu

```
for ( j = mystart; j < myend; j++ )
    A[j] = A[j-1]*2.0;
```

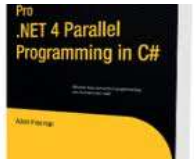
- Giá trị A_{j-1} phải được tính trước giá trị A_j. Như vậy A_j thể hiện sự phụ thuộc dữ liệu vào A_{j-1}. Cơ hội song song không có.

- Nếu task 2 có A_j và task 1 có A_{j-1}, việc tính toán giá trị đúng của A_j cần:
  - Kiến trúc bộ nhớ phân tán - task 2 phải nhận giá trị A_{j-1} từ task 1 sau khi task 1 hoàn thành công việc tính toán của nó.
  - Kiến trúc bộ nhớ chia sẻ - task 2 phải đọc lại A_{j-1} sau khi task 1 cập nhật giá trị A_{j-1}

Ví dụ khác

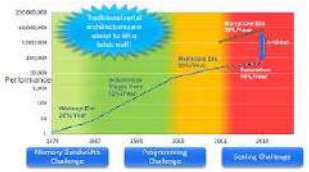
- Task 1 thực hiện các câu lệnh:

  ...
  X = 2
  ...
  Y = X*X



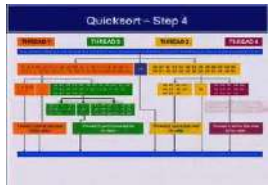
- Task 2 thực hiện lệnh tương tự với X = 4 và tính Y = X*X*X

- Giá trị Y được tính phụ thuộc vào kiến trúc:
  - Distributed Memory: Giá trị X phải được truyền qua lại giữa các task
  - Shared Memory: Task thực hiện sau cùng lưu trữ giá trị của X



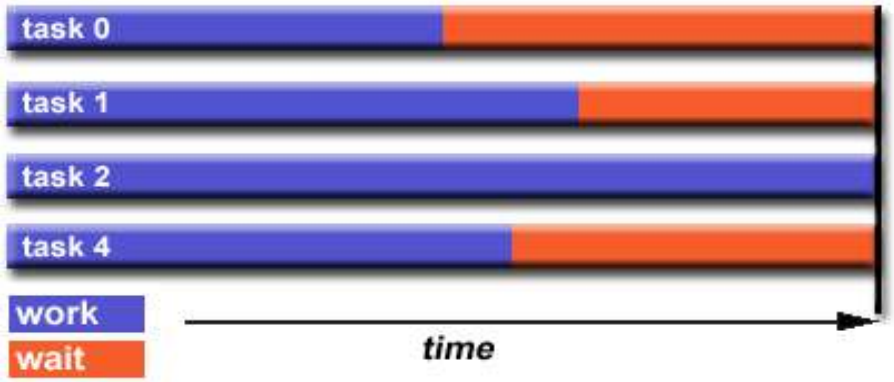
Như vậy

- Làm thế nào để xử lý sự phụ thuộc dữ liệu:
  - Kiến trúc Distributed Memory: giao tiếp dữ liệu cần thiết tại các điểm đồng bộ hóa
  - Kiến trúc Shared Memory: đồng bộ hóa thao tác read/write giữa các task



Load Balancing

- Load balancing (cân bằng tải) đề cập đến việc phân phối công việc trong số các task sao cho tất cả các task giữ sự bận rộn toàn bộ thời gian.
- Điều đó được coi là giảm thời gian chờ đợi (idle time) đến mức cực tiểu.

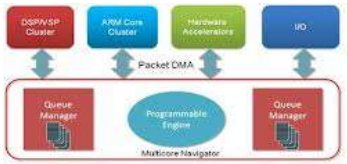


- Vì lý do hiệu năng, nên cân bằng tải rất quan trọng đối với một chương trình song song.
- Chẳng hạn, nếu tất cả các task phải chịu một ngưỡng đồng bộ hó, thì task chậm nhất sẽ quyết định hiệu năng tổng thể.

Làm thế nào để đạt sự cân bằng tải

- Phân chia công việc cho mỗi task là tương đương nhau:
  - Đối với các thao tác trên array/matrix, ở đó mỗi task thực hiện công việc tương tự nhau, khi đó cần phân phối đều dữ liệu trong số các task.
  - Đối với các vòng lặp, công việc được làm trong mỗi vòng làm là tương tự, cần phải phân phối đều các vòng lặp trong các task.

- Nếu có một sự hỗn hợp không đồng nhất của các máy với đặc tính hiệu năng khác nhau được sử dụng, phải bảo đảm dùng cùng một loại công cụ phân tích hiệu năng để xác định sự mất cân bằng tải. Từ đó điều chỉnh công việc cho phù hợp.



Phân công công việc năng động

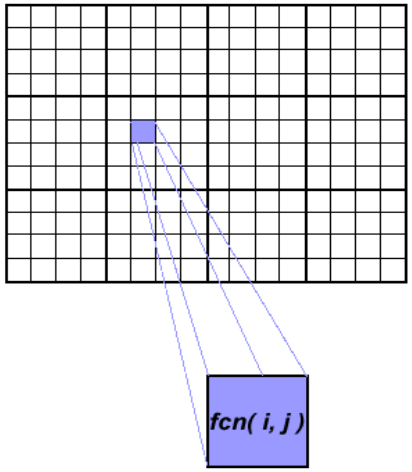
- Có một số bài toán mất cân bằng tải ngay cả khi có sự phân phối đều dữ liệu giữa các task:
  - Ma trận thưa: một vài task có dữ liệu thực sự để làm việc, trong khi một số task khác có chủ yếu là số không.
  - Phương pháp lưới thích nghi (Adaptive grid method): một vài task cần tinh chỉnh lươi, trong khi một số khác thì không.

– Khi số lượng công việc của mỗi task là một đại lượng biến thiên, nên không thể dự đoán được; phải tiếp cận theo hướng lập lịch công việc.
– Khi đó, mỗi task sau khi hoàn thành công việc sẽ đợi để nhận công việc mới.
– Nên cần phải thiết kế một thuật giải mà có thể phát hiện và xử lý sự mất cân bằng tải với những đoạn mạ linh động.

Ví dụ thiết kế chương trình

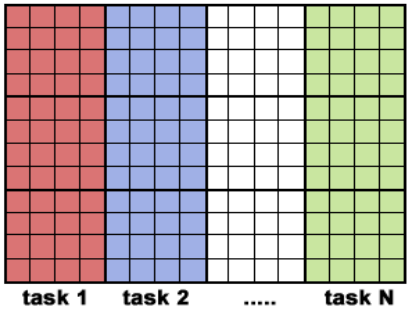
- Với mảng dữ liệu như hình. Tính mỗi phần tử của mảng theo thuật giải tuần tự như:

```
For j = 1 to n
  For i = 1 to n
    a(i,j) = fnc(i,j)
```



- Để song song với giải pháp là chia thành N task như hình.
- Sau khi dữ liệu được phân phối cho các task, mỗi task tính toán với phần dữ liệu do nó sở hữu:

```
For j = start to end
    For i = 1 to n
        a(i,j) = fnc(i,j)
```



- Cài đặt trên mô hình SPMD (Single Program Multiple Data) như sau:

```
if MASTER
  Khởi tạo mảng
  Gửi đến các WORKER thông tin mà nó sở hữu
  Gửi đến các WORKER mảng của nó sở hữu
  Nhận kết quả từ WORKER
```

```
else if WORKER
   Nhận thông tin về mảng từ MASTER
   Nhận từ MASTER phần mảng ban đầu của nó
   For j = my first column,my last column
      For i = 1 to n
         a(i,j) = fcn(i,j)
   Gửi kết quả về cho MASTER
endif
```

Viết bằng MPI